Remarks

This responds to the final Office action mailed October 20, 2006 ["the Action"]. Reconsideration of the application is respectfully requested in view of the following remarks. Claims 1 and 3-39 are pending in the application. No claims have been allowed. No claims are amended. Claims 1, 20, 29, 31, and 36 are independent.

Cited Art

U.S. Patent No. 6,578,090 to Motoyama et al. ("Motoyama") is entitled "System and Method for Interfacing Two Modules Supporting Various Applications."

Amendments

Claims 1, 20, 22, 24-26, and 36-39 have been amended to clarify that they are directed to statutory subject matter under 35 U.S.C. § 101. No new matter has been added.

Claims Should Be Allowable over 35 U.S.C. § 101

Claims 1, 3-19, 20-28, and 36-39 have been rejected by the Action under 35 U.S.C. § 101 as being directed to non-statutory subject matter. Applicants respectfully traverse the rejection.

Claim 1 has been amended to incorporate language of dependent claim 2, which was not rejected under 35 U.S.C. § 101, which recites that "the one or more objects represent type information of a variable in software during compilation of the software." Claim 1, as amended, is directed to statutory subject matter. Additionally, claims 3-19, each of which depend from claim 1 but recite additional language, are thus also allowable under 35 U.S.C. § 101.

Claims 20 and 36 have been amended to recite a "program comprising computer executable instructions for implementing a method for representing type classifications" as well as a plurality of actions. Applicants note the similarity between this language and the langage of claim 29, which was not rejected under 35 U.S.C. § 101. For at least this reason, claims 20 and 36, as amended, are directed to statutory subject matter. Additionally, claims 21-28 and 37-39, each of which depend from either claim 20 or 36 but recite additional language, are thus also allowable under 35 U.S.C. § 101.

For at least these reasons, Applicants respectfully request that the rejection of claims 1, 3-19, 20-28, and 36-39 under 35 U.S.C. § 101 be withdrawn and the claims be allowed.

Claims 1-20 Should be Allowable

The Action rejects claims 1-39 under 35 U.S.C. § 102(e) as being anticipated by Motoyama. Applicants respectfully submit the claims in their present form are allowable over the cited art. For a 102(e) rejection to be proper, the cited art must show each and every element as set forth in a claim. (*See* MPEP § 2131.01.) However, the cited art does not describe each and every element. Accordingly, applicants request that all rejections be withdrawn.

Claim 1, as amended, recites, in part:

> *A method of representing type information via objects of classes* in a class hierarchy, wherein the class hierarchy comprises at least one class and a plurality of sub-classes for representing different type classifications, the method comprising:
>
> *instantiating one or more objects of one or more of the sub-classes of the hierarchy, wherein the one or more sub-classes represent classifications of types . . .*

[Emphasis added.] For example, the Application describes the representation of classifications of types in objects:

> Referring to Appendix A, a number of type representations are defined in a type class hierarchy such that type systems of various languages can be represented by the typed intermediate language. An abstract base class is defined as 'Phx::Type' for all types. The base class can contain, for instance, size information in 'sizekind' for values of specific types. The size may be constant, symbolic or unknown (or variable). The base class can also contain 'typekind' in order to designate type classification. Additionally, an external type can be provided as an abstract type that wraps an externally defined type in order to provide back mapping from the typed intermediate language to the original source code.
>
> Below the base class, a class defined as 'Phx::PtrType' can represent pointer types. Various kinds of pointers can be defined as well. For instance, a managed, garbage collected object pointer (points to the base of a garbage collected object), a managed, garbage collected pointer (points to a location within a garbage collected object), an unmanaged pointer (such as would be found in code written in C++, for instance), and a null pointer.
>
> At the same level in the hierarchy, a class defined as 'Phx::ContainerType' can represent container types, such as types that contain internal members. The internal members can be fields, methods and other types.

> A class defined as 'Phx::FuncType' can represent function types, including any necessary calling conventions, lists of arguments and lists of return types. Also, a class defined as 'Phx::UnmgdArrayType' can represent unmanaged array types. Under 'Phx::ContainerType' in the hierarchy, four more classes can be defined. A class defined as 'Phx::ClassType' can represent class types, a class defined as 'Phx::StructType' can represent struct types, a class defined as 'Phx::InterfaceType' can represent interface types, and a class defined as 'Phx::EnumType' can represent enumerated types. Under 'Phx::ClassType' in the hierarchy, an additional class defined as 'Phx::MgdArrayType' can represent managed array types.

[Application, at page 8, line 8 to page 9, line 6.] The application also gives examples of such representation in Appendix A, at pages 21-37. An outline of the representation is given in code comments in Appendix A:

```
//---------------------------------------------------------------------------//
// Description:
//
//  IR types
//
//
//  Type Class Hierarchy        Description & Primary Properties Introduced
//  --------------------        -----------------------------------------
//  Phx::Type                   - Abstract base class for types
//    Phx::PtrType              - Pointer types
//    Phx::ContainerType        - Container types (types that have members)
//        Phx::ClassType        - Class types
//          Phx::MgdArrayType - Managed array types
//        Phx::StructType       - Struct types
//        Phx::InterfaceType    - Interface types
//        Phx::EnumType         - Enumerated types
//    Phx::FuncType             - Function types
//                              Properties: ArgTypes, ReturnType
//
//    Phx::UnmgdArrayType       - Unmanaged arrays
//                              Properties: Dim, Referent
//
//---------------------------------------------------------------------------
```

Motoyama's description of objects and general object interactions does not teach or suggest "instantiating one or more objects of one or more of the sub-classes of the hierarchy, wherein the one or more sub-classes represent classifications of types" as recited in claim 1. In its rejection of this language of claim 1, the Action cites to Figure 2A and boxes 62 and 64 of

Figure 4, as well as the text accompanying this section. However neither of these sections does talks about the representation of classifications of types.

Figure 2A of Motoyama shows a "class diagram showing a class structure using multiple inheritance to construct various data structures with the functions corresponding to" a "DATA STRUCTURE 7" illustrated in Figure 1. [Motoyama, at column 9, lines 63-66.] However, Motoyama makes it clear that the class structure used for this generation is *specifically designed to avoid representing type information*:

> FIG. 1 illustrates the intent of the system where the APPLICATION PROCESS UNIT 8 must perform various tasks depending upon the type of DATA STRUCTURE 7. *The prior art to the problem is to add the type information in the DATA STRUCTURE 7 . . . The present invention shows that the user of an object-oriented method can eliminate the need of type information.* Moreover, the present invention shows that even when an application changes, the DATA ANALYSIS UNIT 6 does not need to change its processing to the changing application needs.

[Motoyama, at column 9, lines 51-62; emphasis added.] This avoidance from representation of type classification is made further evident from the Abstract:

> There is no need for the first application module to determine which software object type is referenced for execution of the function, as all instantiated software objects are derived from the same abstract class.

Thus, Motoyama teaches away from the representation of classifications of types in the DATA STRUCTURE 7. Motoyama further shows a clear motivation against maintaining type information in the data structures it passes as it discusses how its system seeks to solve the problems of existing systems:

> In programming languages, data types are used to determine the internal representation of data to be used in software programs. . . .
>
> If a user needs to access a software module to perform a function, with the processing results to be used by plural software objects having different structures as data types, *the related art scheme of passing data items along with information concerning the data type of the data items sent involves the disadvantage of writing different executable code to handle every case of data type information for every type of software object in the system.* If the data types are modified, for example, by adding new software object types, or by modification of existing software object types, the code to handle the various data types must be updated or rewritten for every modification made in the system. Such updating or rewriting of the software may be a time consuming and expensive task.

[Motoyama, at column 6, lines 12-42; emphasis added.] Motoyama thus cannot teach or suggest "instantiating one or more objects of one or more of the sub-classes of the hierarchy, wherein the one or more sub-classes represent classifications of types" as recited in claim 1.

Applicants note that Motoyama describes data, at various points, of being of a type. However, the fact that these classes are used to determine that data has a type does not mean that Motoyama teaches or suggests "instantiating one or more objects of one or more of the sub-classes of the hierarchy, wherein the one or more sub-classes represent classifications of types" as in claim 1. For example, Figure 2A of Motoyama shows a class structure with two different derived classes, Derived Class 1 (14) and Derived Class 2 (16). Motoyama additionally says that these classes "correspond[] to a type of DATA STRUCTURE 7" [Application, at column 10, lines 25-26 and 41-42.]

However, this does not mean that Derived Class 1 and Derived Class 2 represent type classifications. Instead, they are simply classes defining particular data structures. Hence, when Motoyama discusses using the Derived Classes to determine a "type," it is clear that this is a structure of data, and not a "classification of types":

> The getAbstractClass 56 inputs data from an input data source, determines a data type for the input data, and sends to the user of the getAbstractClass 56 a pointer to a software object for processing the data type of the input data. . . .
>
> If it is determined that the data is not the end of the input file, then statement 110 determines if the input data is of a type for the Derived Class 114 of FIG. 2A. If it is determined that the input data contains data of a type for the Derived Class 114, then statement 112 of FIG. 2E uses the value of the variable Ptr151 of FIG. 2B to execute the inherited software method setVar120 for an instantiated object of the Application Class 136 of FIG. 2A to set the variable var118 to have a value of the input data.

[Motoyama, at column 15, line 20-37.] Again, it is worth noting that the way the Derived Classes are used in Motoyama is to determine what kind of data structure a given input is based on structural information given in the Derived Class definition. [*See*, Figure 2A.] This is different than the classes described in the Application and claimed in Claim 1, which represent type classifications, such as, for example, the "Enumeration Type" described on page 28 of the Application. Applicants further note that the discussion around blocks 62 and 64 of Figure 4, which were cited in the rejection, while about instantiation of objects, is directed toward the creation of application class objects, and does not discuss data types.

For at least these reasons, Motoyama does not teach or suggest "instantiating one or more objects of one or more of the sub-classes of the hierarchy, wherein the one or more sub-classes represent classifications of types" and thus does not describe each and every element of claim 1. Thus, claim 1, as well as claims 3-19, which depend from claim 1 and each recite patentable matter, are allowable.

With regard to claims 20-39, the Action rejects the claims "for the same reasons as cited in claims 1-17," giving no other indication of the reasons for rejection. [Action, at page 9, paragraph 6.] Thus, for at least the reasons discussed above, Motoyama does not describe each and every element of claims 20-39. Applicants therefore request that the rejections of claims 1 and 3-39 be withdrawn and that the claims be allowed.

Request for Information Disclosure Statement To Be Reviewed

Applicants note that the Action does not include an initialed copy of the Form 1449 which accompanied the Information Disclosure Statements filed on July 27, 2004 and May 3, 2006, as well as a single reference ("Hamilton") from the Information Disclosure Statement filed on July 10, 2006. Applicants request the Examiner provide initialed copies of the indicated Forms 1449.

Request for Interview

In view of the preceding amendments and remarks, Applicants believe the application to be allowable. If any issues remain, however, the Examiner is formally requested to contact the undersigned attorney at (503) 226-7391 prior to issuance of the next communication in order to arrange a telephonic interview. This request is being submitted under MPEP § 713.01, which indicates that an interview may be arranged in advance by a written request.

Conclusion

Claims 1 and 3-39 should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By [illegible]
Stephen A. Wight
Registration No. 37,759